**Lecture1:**

**What is software?**

**Ans**: A **software** is computer programs, procedures, rules and associated documentation and data.   
**What does Software mean?**

* A set of instructions
* A program is an executable code
* Executable computer programs and associated documentation
* A program or set of programs containing instructions which provide desired functionality .

**What is software product?**

**Ans:** Software products may be developed for a particular customer or may be developed for a general market.

Software products may be 2 types:

* + **Generic**- developed to be sold to a range of different customers. ( These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include software for PCs such as databases, word processors, drawing packages and project management tools.)
  + **Customised (Bespoke)** - developed for a single customer according to their specification. ( These are systems which are commissioned by a particular customer. A software contractor develops the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process and air traffic control systems.)

An important difference between these types of software is that, in generic products, the organization that develops the software controls the software specification. For custom products, the specification is usually developed and controlled by the organization that is buying the software. The software developers must work to that specification

**What are the attributes of good software?**

Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

Essential attributes of good software:

1. **Maintainability:** Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
2. **Dependability and security**: Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
3. **Efficiency:** Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
4. **Acceptability:** Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

**Why software engineering is important?**

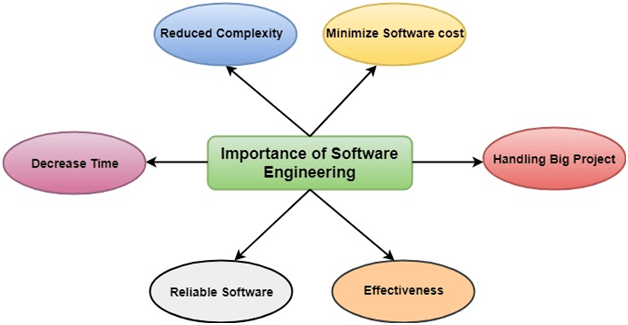
**Ans:**

Software engineering is important for two reasons:

1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

2. It is usually cheaper, in the long run, to use software engineering methods and techniques for professional software systems rather than just write programs as a personal programming project.

Failure to use software engineering method leads to higher costs for testing, quality assurance, and long-term maintenance.

****

**Lecture 2:**

**Software classification:**

**1.System software :** is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications.  
**2.Application software :** is specific to the task it is designed for and can be as simple as a calculator application or as complex as a word processing application. For example, office software suites might include word processing, spreadsheet, database, presentation, and email applications.  
**3.Scientific and engineering software :** satisfies the needs of a scientific or engineering user to perform enterprise specific tasks. Such software is written for specific applications using principles, techniques and formulae specific to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.  
**4.Embedded software :** is computer software, written to control machines or devices that are not typically thought of as computers, commonly known as embedded systems. It is typically specialized for the particular hardware that it runs on and has time and memory constraints.  
**5.WebApps :** Web based application like facebook, daraz, amazon etc.  
**6.Mobile Apps :** Messenger, Whatsapp etc.  
**7.AI Software :** Software that is capable of intelligent behavior. This involves simulating a number of capabilities, including reasoning, learning, problem solving, perception, knowledge representation.

**What’s the difference between software engineering and computer science?**

**Ans:** Essentially, computer science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. Ideally, all of software engineering should be underpinned by theories of computer science, but in reality, this is not the case. Software engineers must often use ad hoc approaches to developing the software. Elegant theories of computer science cannot always be applied to real, complex problems that require a software solution.

**Software engineering:**

Definition: Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

People writing programs for themselves, however, can spend as much time as they wish on the program development. But engineering is about getting results of the required quality within schedule and budget. This often involves making compromises.

In general, software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software. However, engineering is all about selecting the most appropriate method for a set of circumstances

In this definition, there are two key phrases:

1. Engineering discipline: Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work within organizational and financial constraints, and they must look for solutions within these constraints.
2. All aspects of software production: Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development.

**What is the difference between software engineering and system engineering?**

**Ans:** System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

*System engineers* are involved in (overall) system specification, architectural design, integration, and deployment.  
(System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design and system deployment as well as software engineering. System engineers are involved in specifying the system, defining its overall architecture and then integrating the different parts to create the finished system. They are less concerned with the engineering of the system components (hardware, software, etc.). System engineering is an older discipline than software engineering. People have been specifying and assembling complex industrial systems such as aircraft and chemical plants for more than a hundred years. However, as the percentage of software in systems has increased, software engineering techniques such as use-case modelling and configuration management are being used in the systems engineering process.)

**Program vs Software Product:**

* Program is a set of instruction related each other where as Software Product is a collection of program designed for specific task.
* Programs are developed by individuals that means single user where as Software Product are developed by large no of users.
* In program, there is no documentation or lack in proper documentation. In Software Product, Proper documentation and well documented and user manual prepared.
* Development of program is Unplanned, not Systematic etc. but Development of Software Product is well Systematic, organized, planned approach.
* Programs provide Limited functionality and less features where as Software Products provides more functionality as they are big in size (lines of codes) more options and features.

**General issues that affect software:**

There are four related issues that affect many different types of software:

1. Heterogeneity: Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. As well as running on general-purpose computers, software may also have to execute on mobile phones and tablets. You often have to integrate new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to cope with this heterogeneity.
2. Business and social change: Businesses and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software. Many traditional software engineering techniques are time consuming, and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.
3. Security and trust: As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. We have to make sure that malicious users cannot successfully attack our software and that information security is maintained.
4. Scale: Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

**Key challenges facing software engineering?**

Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

1. Legacy systems – (old, valuable systems must be maintained and updated)The majority of software systems which are in use today were developed many years ago yet they perform critical business functions. The legacy challenge is the challenge of maintaining and updating this software in such a way that excessive costs are avoided and essential business services continue to be delivered.
2. Heterogeneity – (systems are distributed and include a mix of hardware and software.) Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and with different kinds of support systems. The heterogeneity challenge is the challenge of developing techniques to build dependable software which is flexible enough to cope with this heterogeneity.
3. Delivery – (there is increasing pressure for faster delivery of software.) Many traditional software engineering techniques are time-consuming. The time they take is required to achieve software quality. However, businesses today must be responsive and change very rapidly. Their supporting software must change equally rapidly. The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.

**challenges** Likely to face 2021:

1. Legacy systems – (old, valuable systems must be maintained and updated)The majority of software systems which are in use today were developed many years ago yet they perform critical business functions. The legacy challenge is the challenge of maintaining and updating this software in such a way that excessive costs are avoided and essential business services continue to be delivered.
2. Rapid technology advancement–

Every technology advancement is a blessing for the IT industry. But at the same time, technology evolving at a phenomenal rate leads to an added pressure for software development professionals to leverage these upcoming technology trends in software product development to gain a cutting edge over competitors and stand out in the market.

1. The trust challenge–

As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. The trust challenge is to develop techniques that demonstrate that software can be trusted by its users.

**Software process:**

The systematic approach that is used in software engineering is sometimes called a software process.

A software process is a sequence of activities that leads to the production of a software product. Four fundamental activities are common to all software processes.

1. Software specification: where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development: where the software is designed and programmed.
3. Software validation: where the software is checked to ensure that it is what the customer requires.
4. Software evolution: where the software is modified to reflect changing customer and market requirements.

These activities are complex activities in themselves, and they include subactivities such as requirements validation, architectural design, and unit testing. Processes also include other activities, such as software configuration management and project planning that support production activities.

**Software Process Model:**

A software process model is a simplified description of a software process that presents one view of that process. Process models may include activities that are part of the software process, software products and the roles of people involved in software engineering. Some examples of the types of software process model that may be produced are:

Software process generic models:

1. The waterfall model: This takes the fundamental process activities of specification, development, validation and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on.
2. Evolutionary development: This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from abstract specifications. This is then refined with customer input to produce a system that satisfies the customer’s needs.
3. Component-based software engineering: This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch.

These three generic process models are widely used in current software engineering practice.

**Q. 🡪two ways in which a software process model might be helpful in identifying possible process improvements:**

1. Incremental development: This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.
2. Integration and configuration: This approach relies on the availability of reusable components or systems. The system development process focuses on configuring these components for use in a new setting and integrating them into a system.

**Difference between Software Process vs Software Process Model:**

1.The systematic approach that is used in software engineering is called a software process whereas,

A software process model is **a simplified description** of a software process that presents one view of that process.

2.The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes where on the other side

In the Software Process Model (waterfall model), they are organized in sequence, while in incremental development they are interleaved

**Internet Software Engineering:**

The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.

Web services allow application functionality to be accessed over the web.

Cloud computing is an approach to the provision of computer services where applications run remotely on the ‘cloud’.

This change in software organization has had a major effect on software engineering for web-based systems. For example:

1. Software reuse has become the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from preexisting software components and systems, often bundled together in a framework.
2. It is now generally recognized that it is impractical to specify all the requirements for such systems in advance. Web-based systems are always developed and delivered incrementally.
3. Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services.

Web-based systems are getting larger and larger, so software engineering techniques that deal with scale and complexity are relevant for these systems.

**Lecture 3:**

**Software engineering ethics:**

Like other engineering disciplines, software engineering is carried out within a social and legal framework that limits the freedom of people working in that area. As a software engineer, you must accept that your job involves wider responsibilities than simply the application of technical skills. You must also behave in an ethical and morally responsible way if you are to be respected as a professional engineer.

It goes without saying that you should uphold normal standards of honesty and integrity.

there are areas where standards of acceptable behavior are not bound by laws but by the more tenuous notion of professional responsibility. Some of these are:

1. **Confidentiality** :You should normally respect the confidentiality of your employers or clients regardless of whether or not a formal confidentiality agreement has been signed.
2. **Competence:** You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. **Intellectual property rights**: You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. Computer misuse: You should not use your technical skills to misuse other people’s computers. Computer misuse ranges from relatively trivial (game playing on an employer’s machine) to extremely serious (dissemination of viruses or other malware).

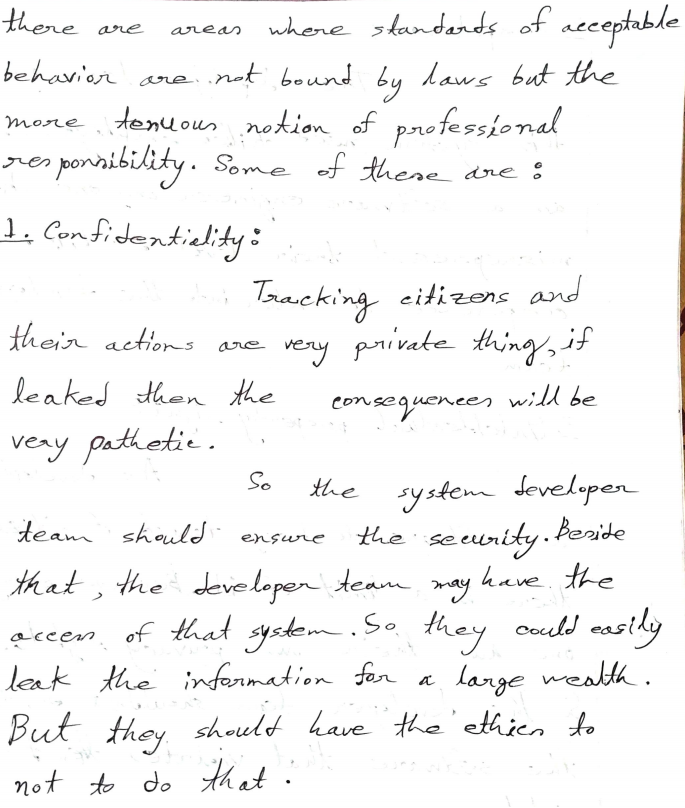
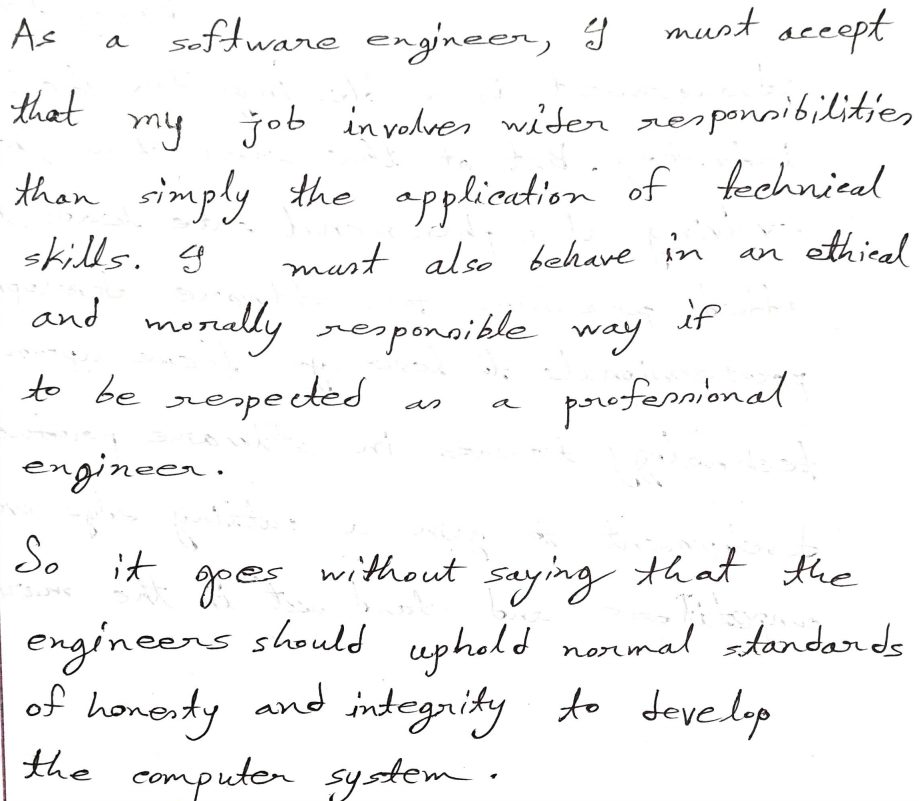
**In short Software engineering ethics:**

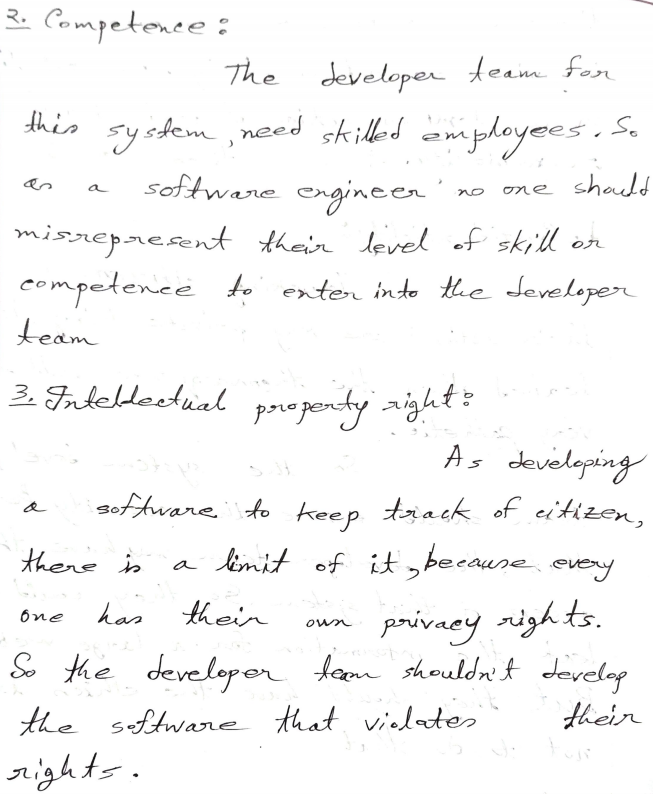
Software engineering involves wider responsibilities than simply the application of technical skills.

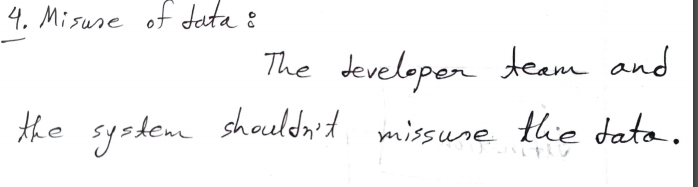
Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.

Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

**Q. To help counter terrorism, many countries are planning the development of computer systems that track large numbers of their citizens and their actions. Clearly this has privacy implications. Discuss the ethics of developing this type of system?**







**What is System:**

A system is a purposeful collection of interrelated components of different kinds that work together to deliver a set of services to the system owner and its users.

* A system may include software, mechanical, electrical and electronic hardware and be operated by people.
* System components are dependent on other system components.

**System categories**

* 1. **Technical computer-based systems:** are systems that include hardware and software components but not procedures and processes. Examples of technical systems include televisions, mobile phones and most personal computer software. Individuals and organisations use technical systems for some purpose but knowledge of this purpose is not part of the system. For example, the word processor I am using is not aware that is it being used to write a book.
  2. **Socio-technical systems:** include one or more technical systems but, crucially, also include knowledge of how the system should be used to achieve some broader objective. This means that these systems have defined operational processes, include people (the operators) as inherent parts of the system, are governed by organisational policies and rules and may be affected by external constraints such as national laws and regulatory policies. For example, this book was created through a socio-technical publishing system that includes various processes and technical systems.

**There are some example case studies: see those.**

**Socio-technical systems:**

Socio-technical system characteristics

1. Emergent properties:  
   Properties of the system of a whole that depend on the system components and their relationships. Security and dependability are examples of important emergent system properties.

(They have emergent properties that are properties of the system as a whole rather than associated with individual parts of the system. Emergent properties depend on both the system components and the relationships between them. As this is so complex, the emergent properties can only be evaluated once the system has been assembled.)

1. Non-deterministic:  
   They do not always produce the same output when presented with the same input because the system's behavior is partially dependent on human operators.  
   ( They are often nondeterministic. This means that, when presented with a specific input, they may not always produce the same output. The system’s behaviour depends on the human operators, and people do not always react in the same way. Furthermore, use of the system may create new relationships between the system components and hence change its emergent behaviour.)
2. Complex relationships with organizational objectives:  
   The extent to which the system supports organizational objectives does not just depend on the system itself.

(The extent to which the system supports organisational objectives does not just depend on the system itself. It also depends on the stability of these objectives, the relationships and conflicts between organisational objectives and how people in the organisation interpret these objectives. New management may reinterpret the organisational objective that a system is designed to support, and a ‘successful’ system may then become a ‘failure’.)

**Why it is important for software engineers to know about systems engineering and to be active participants in systems engineering processes?**

**Ans:**  
1. Software is now the dominant element in all enterprise systems, yet many senior decision makers in organizations have a limited understanding of software. Software engineers have to play a more active part in high-level systems decision making if the system software is to be dependable and developed on time and to budget.

2. As a software engineer, it helps if you have a broader awareness of how software interacts with other hardware and software systems, and the human, social, and organizational factors that affect the ways in which software is used. This knowledge helps you understand the limits of software and to design better software systems.

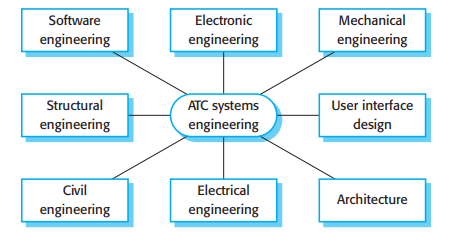
**Systems engineering:**

Systems engineering is the activity of specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.

Systems engineers are not just concerned with software but also with hardware and the system’s interactions with users and its environment.

Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used.

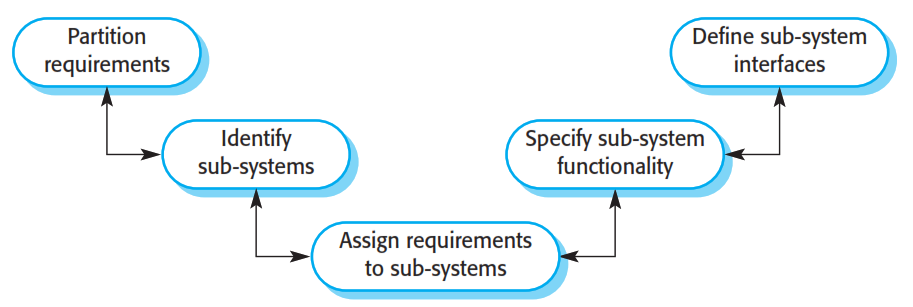
The phases of the systems engineering process are,



Software engineering is a small part of system engineering.

How could we design the system? :

System design:



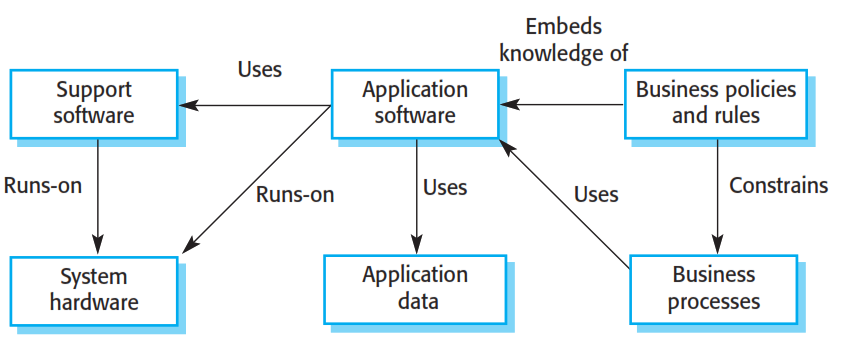
System design (Figure 2.4) is concerned with how the system functionality is to be provided by the components of the system. The activities involved in this process are:

1. Partition requirements You analyse the requirements and organise them into related groups. There are usually several possible partitioning options, and you may suggest a number of alternatives at this stage of the process.
2. Identify sub-systems You should identify sub-systems that can individually or collectively meet the requirements. Groups of requirements are usually related to sub-systems, so this activity and requirements partitioning may be amalgamated. However, sub-system identification may also be influenced by other organisational or environmental factors.
3. Assign requirements to sub-systems You assign the requirements to subsystems. In principle, this should be straightforward if the requirements partitioning is used to drive the sub-system identification. In practice, there is never a clean match between requirements partitions and identified sub-systems. Limitations of externally purchased sub-systems may mean that you have to change the requirements to accommodate these constraints.
4. Specify sub-system functionality You should specify the specific functions provided by each sub-system. This may be seen as part of the system design phase or, if the sub-system is a software system, part of the requirements specification activity for that system. You should also try to identify relationships between sub-systems at this stage.
5. Define sub-system interfaces You define the interfaces that are provided and required by each sub-system. Once these interfaces have been agreed upon, it becomes possible to develop these sub-systems in parallel.

**Legacy systems:**

Legacy systems are socio-technical computer-based systems that have been developed in the past, often using older or obsolete technology. These systems include not only hardware and software but also legacy processes and procedures—old ways of doing things that are difficult to change because they rely on legacy software.

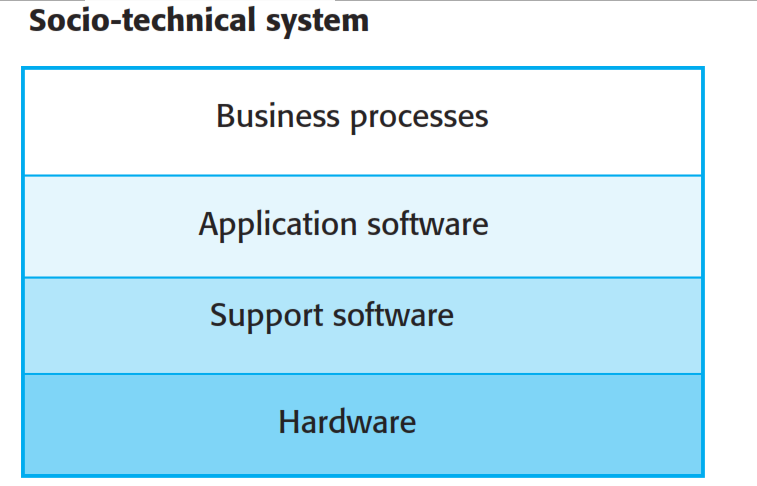
(Legacy systems are often business-critical systems. They are maintained because it is too risky to replace them. For example, for most banks the customer accounting system was one of their earliest systems. Organizational policies and procedures may rely on this system. If the bank were to scrap and replace the customer accounting software (which may run on expensive mainframe hardware) then there would be a serious business risk if the replacement system didn’t work properly. Furthermore, existing procedures would have to change, and this may upset the people in the organization and cause difficulties with the bank’s auditors.)

**Question: illustrates the logical parts of a legacy system and their relationships/Legacy system components:** ****

1. System hardware In many cases, legacy systems have been written for mainframe hardware that is no longer available, that is expensive to maintain and that may not be compatible with current organisational IT purchasing policies.
2. Support software The legacy system may rely on a range of support software from the operating system and utilities provided by the hardware manufacturer through to the compilers used for system development. Again, these may be obsolete and no longer supported by their original providers.
3. Application software The application system that provides the business services is usually composed of a number of separate programs that have been developed at different times. Sometimes the term legacy system means this application software system rather than the entire system.
4. Application data These are the data that are processed by the application system. In many legacy systems, an immense volume of data has accumulated over the lifetime of the system. This data may be inconsistent and may be duplicated in several files.
5. Business processes These are processes that are used in the business to achieve some business objective. An example of a business process in an insurance company would be issuing an insurance policy; in a manufacturing company, a business process would be accepting an order for products and setting up the associated manufacturing process. Business processes may be designed around a legacy system and constrained by the functionality that it provides.
6. Business policies and rules These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

**Layered model of legacy system: show the relation ship:**

An alternative way of looking at these components of a legacy system is as a series of layers. Each layer depends on the layer immediately below it and interfaces with that layer. If interfaces are maintained, then you should be able to make changes within a layer without affecting either of the adjacent layers.



In practice, this simple encapsulation rarely works, and changes to one layer of the system may require consequent changes to layers that are both above and below the changed level. The reasons for this are:

1. Changing one layer in the system may introduce new facilities, and higher layers in the system may then be changed to take advantage of these facilities. For example, a new database introduced at the support software layer may include

facilities to access the data through a web browser, and business processes may be modified to take advantage of this facility.

2.Changing the software may slow the system down so that new hardware is needed to improve the system performance. The increase in performance from the new hardware may then mean that further software changes which were previously impractical become possible.

3. It is often impossible to maintain hardware interfaces, especially if a radical change to a new type of hardware is proposed. For example, if a company moves from mainframe hardware to client-server systems these usually have different operating systems. Major changes to the application software may therefore be required.

**Chapter 3:**

**Critical System:**

It is the systems that fail to deliver the expected objective in which any system failure can result in economic losses, physical damage or human life loss.

* **Safety-critical systems**
  + **Failure** results in loss of life, damage to the environment;
  + **Chemical plant protection** system
* **Mission-critical systems**
  + **Failure** results in failure of some goal-directed activity;
  + **Spacecraft navigation** system
* **Business-critical systems**
  + **Failure** results in high economic losses;
  + Customer **accounting system** in a bank

**System Dependability:** The dependability of a computer system is a property of the system that reflects the user’s degree of trust in the system.

Principal **dimensions of dependability** are:

* + Availability
  + Reliability
  + Safety
  + Security



There are four principal dimensions to dependability,

1. Availability Informally, the availability of a system is the probability that it will be up and running and able to deliver useful services at any given time.
2. Reliability Informally, the reliability of a system is the probability, over a given period of time, that the system will correctly deliver services as expected by the user.
3. Safety Informally, the safety of a system is a judgement of how likely it is that the system will cause damage to people or its environment.
4. Security Informally, the security of a system is a judgement of how likely it is that the system can resist accidental or deliberate intrusions.

**Other dependability properties**

* **Repairability**

Reflects the extent to which the system can be repaired in the event of a failure.

* **Maintainability**

Reflects the extent to which the system can be adapted to new requirements.

* **Survivability**

Reflects the extent to which the system can deliver services whilst under hostile attack.

* **Error tolerance**

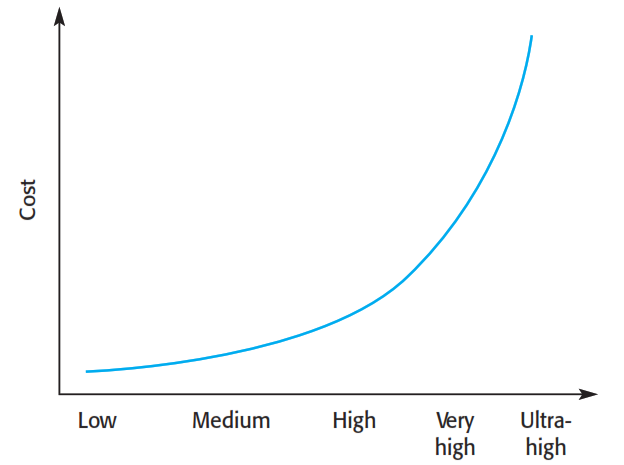
Reflects the extent to which user input errors can be avoided and tolerated.

**Dependability Costs Curve :**

Dependability costs tend to increase exponentially as increasing levels of dependability are required

There are two reasons for this

* + The use of **more expensive development techniques and hardware** that are required to achieve the **higher levels of dependability**
  + The increased testing and system validation that is required to **convince the system client** that the required levels of dependability have been achieved



requirements

Availability, reliability, safety, security

Availability, reliability, safety

Availability, reliability

Availability, only

Basically,

When user requirements are becoming high cost becomes high (exponentially )

**A example of safety critical system and system dependability:\*\*\*\*\*\*\***

(most critical systems are socio-technical systems where people monitor and control the operation of computer-based systems.

There are three ‘system components’ where critical systems failures may occur:

1. System hardware may fail because of mistakes in its design, because components fail as a result of manufacturing errors, or because the components have reached the end of their natural life.

2. System software may fail because of mistakes in its specification, design or implementation.

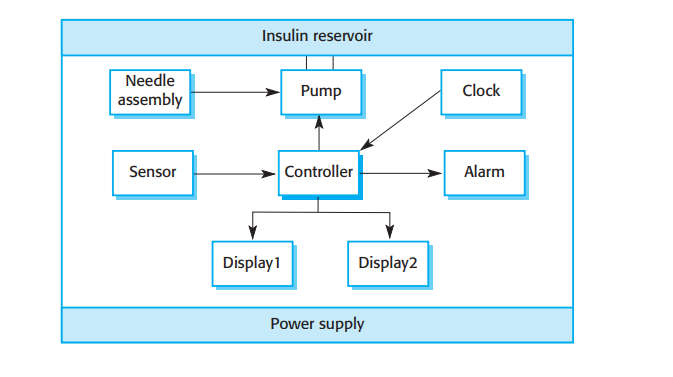
3. Human operators of the system may fail to operate the system correctly. As hardware and software have become more reliable, failures in operation are now probably the largest single cause of system failures.)

Diabetes is a relatively common condition where the human pancreas is unable to produce sufficient quantities of a hormone called insulin.

The problem with this treatment is that the level of insulin in the blood does not just depend on the blood glucose level but is a function of the time when the insulin injection was taken.

Current advances in developing miniaturised sensors have meant that it is now possible to develop automated insulin delivery systems.

Figure is optional:



There are two high-level dependability requirements for this insulin pump system:

1. The system shall be available to deliver insulin when required.

2. The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.

Failure of the system could, in principle, cause excessive doses of insulin to be delivered and this could threaten the life of the user. It is particularly important that overdoses of insulin should not occur.

**Lecture 4:**

**Chapter 4:**

**What is SDLC?**

SDLC stands for **Software Development Lifecycle**. **SDLC** is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations.

**Why SDLC?**

* It offers a basis for project planning, scheduling, and estimating
* Provides a framework for a standard set of activities and deliverables
* It is a mechanism for project tracking and control
* Increases visibility of project planning to all involved stakeholders of the development process
* Increased and enhance development speed
* Improved client relations
* Helps you to decrease project risk and project management plan overhead

**SDLC Phases**

The entire SDLC process divided into the following stages:



### Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

### Phase 2: Feasibility study:

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

**There are mainly five types of feasibilities checks:**

* **Economic:**Can we complete the project within the budget or not?
* **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
* **Operation feasibility:** Can we create operations which is expected by the client?
* **Technical:** Need to check whether the current computer system can support the software
* **Schedule:** Decide that the project can be completed within the given schedule or not.

### Phase 3: Design:

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

* Brief description and name of each module
* An outline about the functionality of every module
* Interface relationship and dependencies between modules
* Database tables identified along with their key elements
* Complete architecture diagrams along with technology details

Low-Level Design(LLD)

* Functional logic of the modules
* Database tables, which include type and size
* Complete detail of the interface
* Addresses all types of dependency issues
* Listing of error messages
* Complete input and outputs for every module

### Phase 4: Coding:

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

### Phase 5: Testing:

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

### Phase 6: Installation/Deployment:

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

### Phase 7: Maintenance:

Once the system is deployed, and customers start using the developed system, following 3 activities occur

* Bug fixing - bugs are reported because of some scenarios which are not tested at all
* Upgrade - Upgrading the application to the newer versions of the Software
* Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

**Software Requirement Specification:**

Software requirement specification(SRS) is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include **ER diagrams**, **data flow diagrams (DFDs)**, function decomposition diagrams (FDDs), **data dictionaries**, etc.

**Data Flow Diagrams:**

Data Flow Diagrams (DFDs) are used widely **for modeling** the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

**Data Dictionaries:**

Data Dictionaries are simply repositories to **store information about all data items** defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.

**Entity-Relationship Diagrams:**

Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram***." It is a detailed **logical representation of the data** for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

**Software Process Model:**

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

A software process model is a simplified description of a software process that presents one view of that process. Process models may include activities that are part of the software process, software products and the roles of people involved in software engineering. Some examples of the types of software process model that may be produced are:

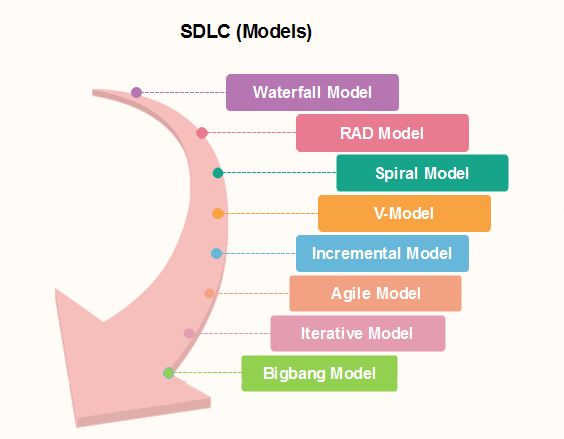
Software process generic models:

1. **The waterfall model:** This takes the fundamental process activities of specification, development, validation and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on.
2. **Evolutionary development:** This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from abstract specifications. This is then refined with customer input to produce a system that satisfies the customer’s needs.
3. **Component-based software engineering:** This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch.

These three generic process models are widely used in current software engineering practice.

Process descriptions may also include:

* **Products,** which are the outcomes of a process activity;
* **Roles,** which reflect the responsibilities of the people involved in the process;
* **Pre- and post-conditions,** which are statements that are true before and after a process activity has been enacted or a product produced.



**Waterfall** model, **spiral** model and **agile** model is most important.

**1.What is The Waterfall Model? (Iterative)**

* Also called **classic software life cycle** or **sequential model or linear-sequential life cycle model (5 component model)**
* Process activities (phases/stages) are **clearly separated**
* The following phase should not start until the previous phase has finished.
* When requirements are frequently changing, we cannot use this model.

2.1.Waterfall-model.eps

1. Requirements analysis and definition The system’s services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

2. System and software design The systems design process partitions the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

3. Implementation and unit testing During this stage, the software design is realised as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

4. Integration and system testing The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

5. Operation and maintenance Normally (although not necessarily) this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system’s services as new requirements are discovered.

**When to use SDLC Waterfall Model**

Waterfall model can be used when

* Requirements are not changing frequently
* Can be used also for parts of larger software systems
* Requirement is clear
* Environment is stable
* Technology and tools used are not dynamic and is stable
* Resources are available and trained

**Drawback of the waterfall model**

* Difficult to incorporate change requests.
* Incremental delivery not supported.
* Overlapping of phases not supported.
* Risk handling not supported.
* Limited customer interactions.

**Lecture 6:**

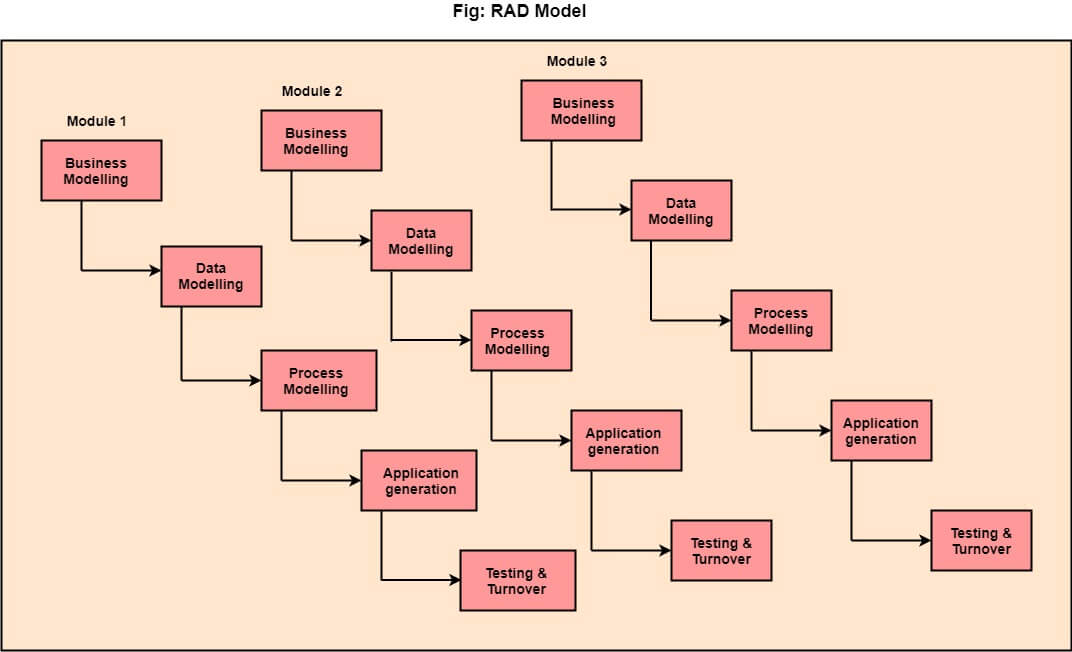
**2.Rapid application development model (RAD):**

Rapid application development model (RAD) is a linear sequential software development process model that emphasizes a concise development cycle using an element-based construction approach.

Rapid Application Development process is an **adoption of the waterfall model**, it targets at developing software in a **short span of time.**

RAD model has following phases

* + Business Modeling
  + Data Modeling
  + Process Modeling
  + Application Generation
  + Testing and Turnover
* **1.Business Modelling:** The information flow among business functions is defined by **answering questions** like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.
* **2. Data Modelling:** The data collected from business modeling is refined into **a set of data objects (entities)** that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.
* **3. Process Modelling:** The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
* **4. Application Generation:** Automated tools are used to facilitate construction of the software; even they use the fourth-generation language (4GL) technologies.
* **5. Testing & Turnover:** Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.



**When to use RAD Model?**

* When the system should need to create the project that modularizes in a short span time (2-3 months).
* It is also suitable for projects where requirements can be modularized and reusable components are also available for development.
* The model can also be used when already existing system components can be used in developing a new system with minimum changes.
* This model can only be used if the teams consist of domain experts. This is because relevant knowledge and ability to use powerful techniques is a necessity.
* The model should be chosen when the budget permits the use of automated tools and techniques required.

**Advantage of RAD Model**

* Use of reusable components helps to reduce the cycle time of the project.
* Feedback from the customer is available at initial stages.
* Reduced costs as fewer developers are required.
* Use of powerful development tools results in better quality products in comparatively shorter time spans.
* The progress and development of the project can be measured through the various stages.
* It is easier to accommodate changing requirements due to the short iteration time spans.

**Disadvantage of RAD Model**

* It required highly skilled designers.
* All application is not compatible with RAD.
* For smaller projects, we cannot use the RAD model.
* On the high technical risk, it's not suitable.
* Required user involvement.

**3.Spiral Model: \*\*\*\***

What is Spiral Model?

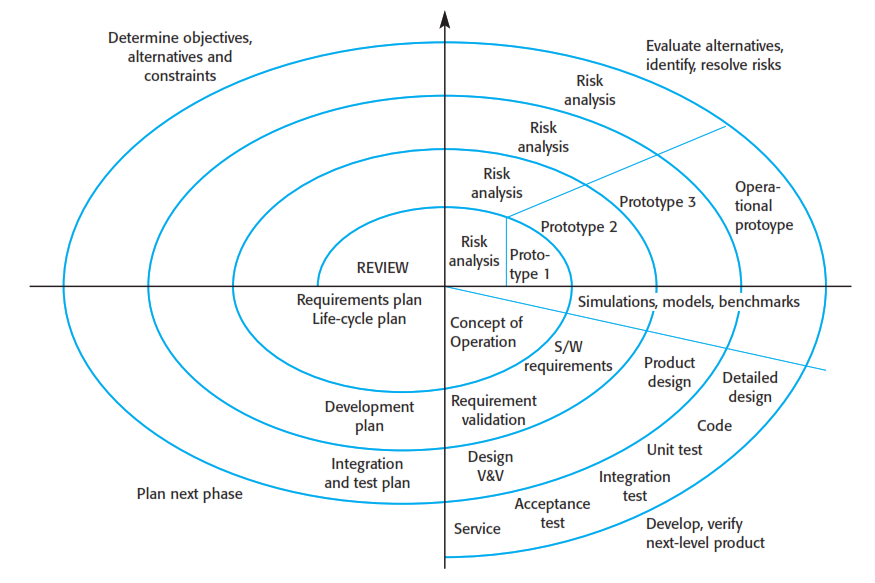
**Spiral model** is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**.

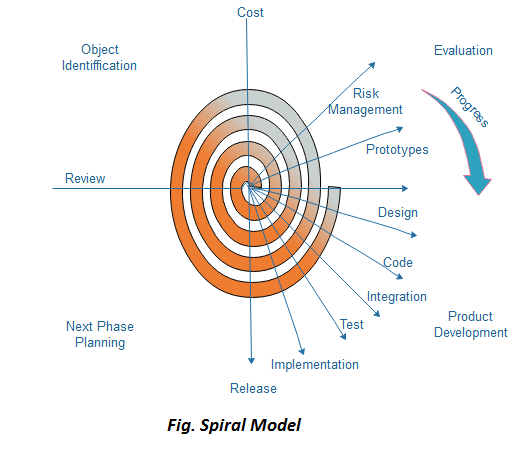
Each phase in spiral model begins with a design goal (review) and ends with the client reviewing the progress.

**It implements the potential for rapid development** of new versions of the software.

The development team in Spiral-SDLC model starts with a small set of requirement and goes through each development phase for those set of requirements.

The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.





**Spiral Model phase:**

**Objective setting**

Specific objectives for the phase are identified.

**Risk assessment and reduction**

Risks are assessed and activities put in place to reduce the key risks.

**Development and validation**

A development model for the system is chosen which can be any of the generic models.

**Planning**

The project is reviewed and the next phase of the spiral is planned.

**Why Spiral Model is called Meta Model ?**

The Spiral model is called as a Meta Model because it subsumes all the other SDLC models.

**When to use Spiral Model?**

* + When deliverance is required to be **frequent**.
  + When the project is **large**
  + When requirements are unclear and complex
  + When changes may require at any time
  + Large and high budget projects

**Advantages**

* + Risk Handling: The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
  + Good for large projects:  It is recommended to use the Spiral Model in large and complex projects.
  + Flexibility in Requirements:  Change requests in the Requirements at later phase can be incorporated accurately by using this model.
  + Customer Satisfaction: Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

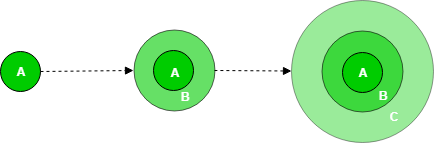
**Disadvantages**

* + Complex: The Spiral Model is much more complex than other SDLC models.
  + Expensive:  Spiral Model is not suitable for small projects as it is expensive.
  + Too much dependable on Risk Analysis: he successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced expertise, it is going to be a failure to develop a project using this model.
  + Difficulty in time management: As the number of phases is unknown at the start of the project, so time estimation is very difficult.

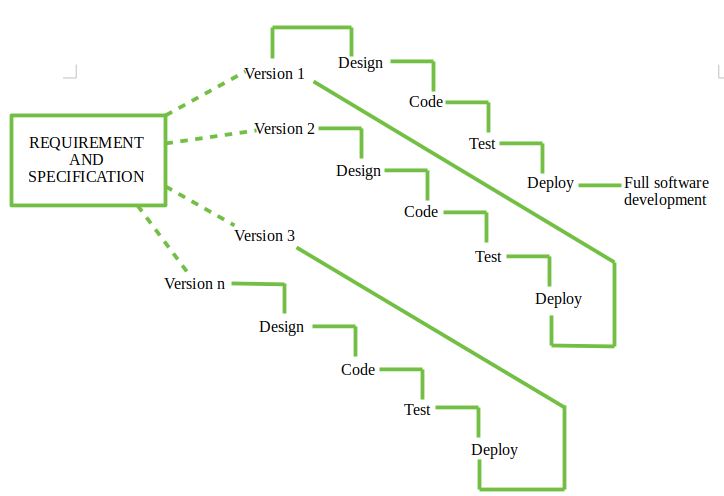
**4.Incremental Model: ( lecture 6)**

**Incremental process model** is also know as **Successive version model**.

First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.



Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle.



**When we use the Incremental Model?**

* + Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
  + When Requirements are known up-front.
  + When Projects having lengthy developments schedules.
  + Projects with new Technology.

**Advantage**

* + Errors are easy to be recognized.
  + Easier to test and debug
  + More flexible.
  + Simple to manage risk because it handled during its iteration.
  + The Client gets important functionality early.

**Disadvantage**

* + Need for good planning
  + Total Cost is high.
  + Well defined module interfaces are needed.

5.V-model:

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model.

It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase.

* The left side of the model is Software Development Life Cycle - **SDLC**
* The right side of the model is Software Test Life Cycle - **STLC**
* The entire figure looks like a V, hence the name **V – model**
* V-Model referred to as the Verification and Validation Model.
* Process executes in a sequential manner in V-shape.
* It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase.
* The next phase starts only after completion of the previous phase.



**Design Phase:**

* **Requirement Analysis:** This phase contains detailed communication with the customer to understand their requirements and expectations. This stage is known as Requirement Gathering.
* **System Design:** This phase contains the system design and the complete hardware and communication setup for developing product.
* **Architectural Design:** System design is broken down further into modules taking up different functionalities. The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood.
* **Module Design:** In this phase the system breaks down into small modules. The detailed design of modules is specified, also known as Low-Level Design (LLD).

**Testing Phases:**

* Unit Testing: Unit Test Plans are developed during module design phase. These Unit Test Plans are executed to eliminate bugs at code or unit level.
* Integration testing: After completion of unit testing Integration testing is performed. In integration testing, the modules are integrated and the system is tested. Integration testing is performed on the Architecture design phase. This test verifies the communication of modules among themselves.
* System Testing: System testing test the complete application with its functionality, inter dependency, and communication. It tests the functional and non-functional requirements of the developed application.
* User Acceptance Testing (UAT): UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets user’s requirement and system is ready for use in real world.

**When to use V-model:**

* Where requirements are clearly defined and fixed.
* The V-Model is used when ample technical resources are available with technical expertise.
* Proactive defect tracking – that is defects are found at early stage.

**Advantages:**

* V-Model is used for small projects where project requirements are clear.
* Simple and easy to understand and use.
* Probability of building an error-free and good quality product.

**Disadvantages:**

* It is not a good for complex and object-oriented projects.
* It is not suitable for projects where requirements are **not clear and contains** high risk of changing.
* It does not easily handle concurrent events.

**Lecture 7:**

**6.Agile Model:**

**Agile model** is a combination of **iterative** and **incremental** process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

It was specially designed to curate the needs of the **rapidly changing environment** by embracing the idea of incremental development and develop the actual final product.

Agile working process:

Agile Methods break the product into **small incremental builds**. These builds are provided in iterations.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

At the end of the iteration a working product is displayed to the customer and important stakeholders.

Agile model believes that every project needs to be **handled differently** and the existing methods need to be tailored to best suit the project requirements.

In agile the tasks are divided to small time frames to deliver specific features for a release.

**Phases of Agile Model:**

**Following are the phases in the Agile model are as follows:**

* + **Requirements gathering**
  + **Design the requirements**
  + **Construction/ iteration**
  + **Testing/ Quality assurance (extreme Programming (XP))**
  + **Deployment**
  + **Feedback**



**The agile software development emphasizes on four core values**.

* 1. **Individuals and interactions** over processes and tools.
  2. **Working software** over comprehensive documentation.
  3. **Customer collaboration** over contract negotiation.
  4. **Responding to change** over following a plan.

**Principles of Agile model:**

1. Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face–to–face conversation**.

**When to use the Agile Model?**

* When frequent changes are required.
* When a highly qualified and experienced team is available.
* When a customer is ready to have a meeting with a software team all the time.
* When project size is small.

**Advantage :**

* Frequent Delivery
* Face-to-Face Communication with clients.
* Efficient design and fulfils the business requirement.
* Anytime changes are acceptable.
* It reduces total development time.

**Disadvantages :**

* Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
* There is a very high individual dependency, since there is minimum documentation generated.
* Transfer of technology to new team members may be quite challenging due to lack of documentation.

**🡪Difference between Agile and Waterfall Model**

**Key Difference**

1. Waterfall is a **Liner Sequential** Life Cycle Model whereas Agile is a **continuous iteration** of development and testing in the software development process.
2. Agile methodology is known for its flexibility whereas Waterfall is a structured software development methodology.
3. Agile follows an incremental approach whereas the Waterfall methodology is a sequential design process.
4. Agile performs testing concurrently with software development whereas in Waterfall methodology testing comes after the “Build” phase.
5. Agile allows changes in project development requirement whereas Waterfall has no scope of changing the requirements once the project development starts.

Now, testing in agile,

**eXtreme Programming (XP)**

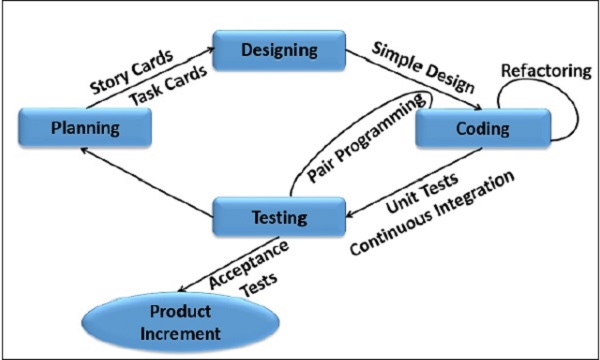
**Extreme programming (XP)** is a Software Development Methodology which is intended to improve software quality and responsiveness to changing customer requirements.

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

XP is the most specific of the **agile frameworks** regarding appropriate engineering practices for software development.

Extreme Programming technique is very helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system.

It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented. The XP develops software keeping customer in the target.



1

**Why is it called “eXtreme”?**

* eXtreme Programming takes the effective principles and practices to extreme levels.
* Code reviews are effective as the code is reviewed all the time.
* Testing is effective as there is continuous regression and testing.
* Design is effective as everybody needs to do refactoring daily.
* Integration testing is important as integrate and test several times a day.
* Short iterations are effective as the planning game for release planning and iteration planning.

Testing needs some planing. Like if we need to test a camera we need to plan first how we would test it. We can test in portrait or landscape or we can do some automation that will test all the functionalities of the camera. Based on planing we need to design the test and write codes to do unit tests. Then we implement the testing code and methods in test section where we actually test the product. Now while testing lets assume the code to test camera flash isn't working so this bug info is sent to coding section the refactor the code and then again the flash is tested. So testing and refactoring happens parallelly. If the product passes testing then it's accepted otherwise if there's a bug the bug report will be sent to coding section or if the product fails the test then we can goto planing considering that there was some mistake in planing. Testing happens in each iteration.

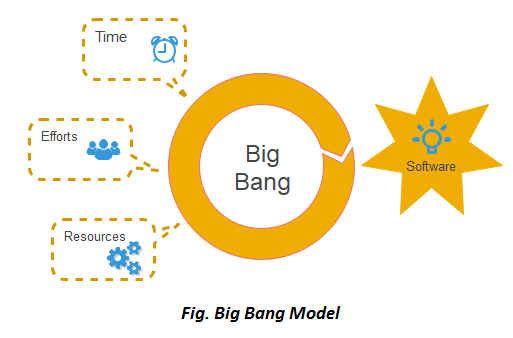
**7.Big Bang Model:**

**When to use Big Bang Model?**

This model is required when this project is small like an academic project or a practical project. This method is also used when the size of the developer team is small and when requirements are not defined, and the release date is not confirmed or given by the customer.

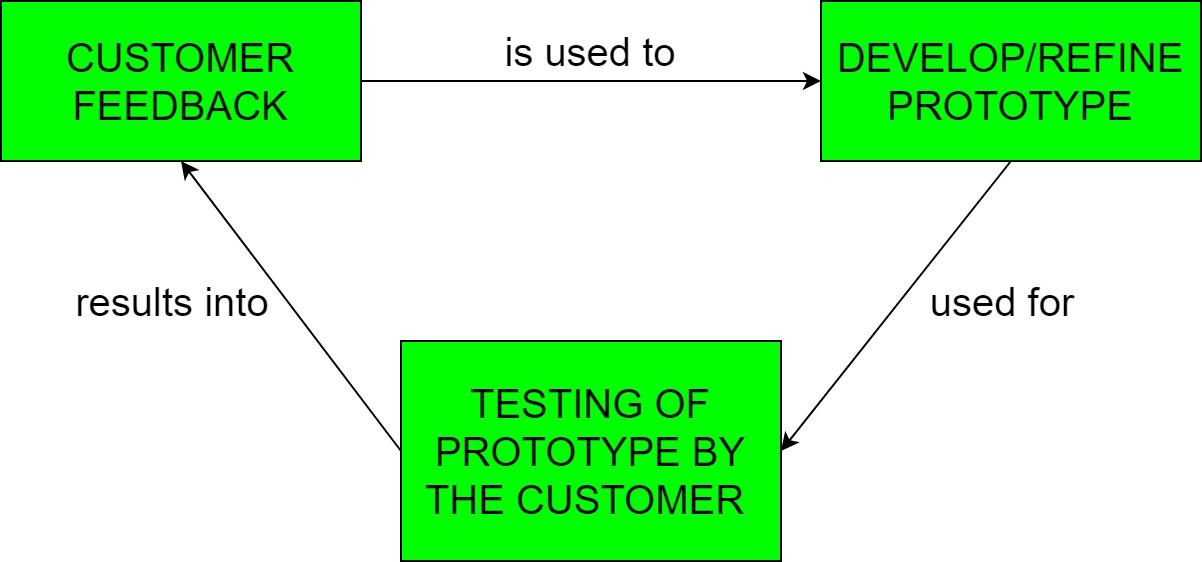
**Advantages:**

1. There is no planning required.
2. Simple Model.
3. Few resources required.
4. Easy to manage.
5. Flexible for developers.



**8.Prototyping Model:**

* In Software Engineering, Prototype methodology is a software development model in which a prototype is built, test and then reworked when needed until an acceptable prototype is achieved.
* Regular meetings are essential to keep the project on time and avoid costly delays in prototyping approach.
* Missing functionality can be identified
* Prototyping may encourage excessive change requests.



**There are 2 approaches for this model:**

1. Rapid Throwaway Prototyping
2. Evolutionary Prototyping

**When to use**

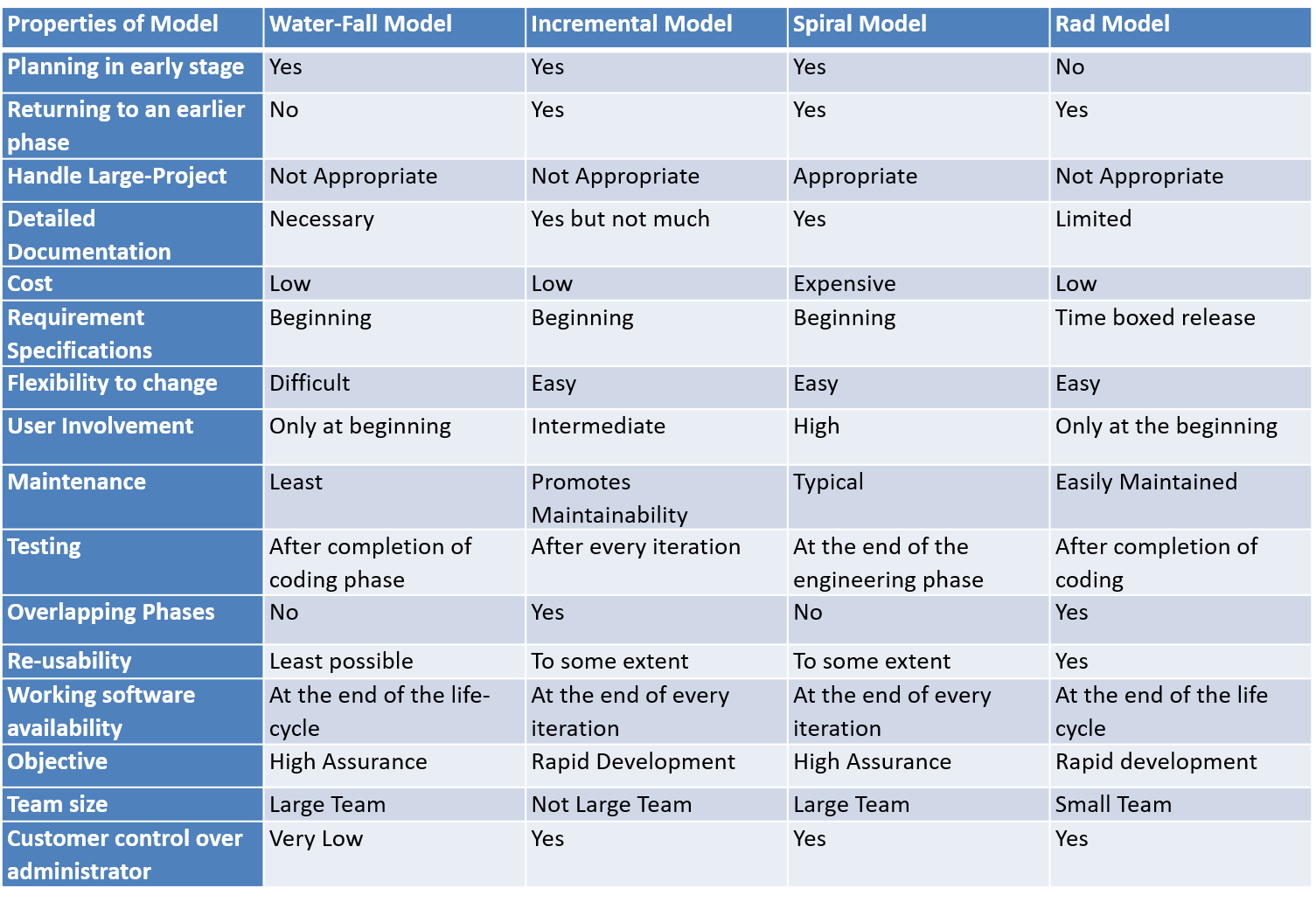
The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable. It can also be used if requirements are changing quickly. This model can be successfully used for developing user interfaces, high technology software-intensive systems, and systems with complex algorithms and interfaces. It is also a very good choice to demonstrate the technical feasibility of the product.

**Advantage of Prototype Model**

* Reduce the risk of incorrect user requirement
* Good where requirement are changing/uncommitted
* Regular visible process aids management
* Support early product marketing
* Reduce Maintenance cost.
* Errors can be detected much earlier as the system is made side by side.

**Disadvantage of Prototype Model**

* An unstable/badly implemented prototype often becomes the final product.
* Require extensive customer collaboration
* Difficult to know how long the project will last.
* Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
* Prototyping tools are expensive.
* Special tools & techniques are required to build a prototype.
* It is a time-consuming process.



**Comparison of Various SDLC Models**

**Software Evolution:**

**Software evolution** is concerned with modifying existing software systems to meet new requirements. This is becoming the normal approach to software development for small and medium-sized systems.



**Rational Unified Process:**

* The **Rational Unified Process (RUP)** is an example of a modern process model derived from the work on the UML and associated process.
* Normally described from 3 perspectives
  + A dynamic perspective that shows phases over time;
  + A static perspective that shows process activities;
  + A practice perspective that suggests good practice.



* **Inception**
  + Establish the business case for the system.
* **Elaboration**
  + Develop an understanding of the problem domain and the system architecture.
* **Construction**
  + System design, programming and testing.
* **Transition**
  + Deploy the system in its operating environment.

**Rational Unified Process:**

**Six fundamental best practices are recommended**:

* Develop software iteratively
* Manage requirements
* Use component-based architectures
* Visually model software
* Verify software quality
* Control changes to software

**Computer-Aided Software Engineering**

**CASE classification**

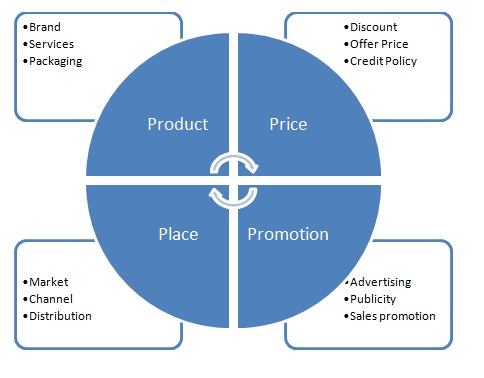
* Classification helps us understand the different types of CASE tools and their support for process activities.
* **Functional perspective**
  + Tools are classified according to their specific function.
* **Process perspective**
  + Tools are classified according to process activities that are supported.
* **Integration perspective**
  + Tools are classified according to their organisation into integrated units.

**C ase tools:**

**Book🡪 88 page, Chapter 4,**

**Lecture 8:**

**4Ps in Marketing**:



Before you can sell a product, you need a marketing strategy to build connections between the product and your target customer. The 4 P’s – product, place, price, and promotion – are a common marketing mix that can give you an [integrated marketing approach](https://www.mindtools.com/pages/article/newSTR_94.htm). Each of the four basic marketing elements has a different impact on how you market a product.

Promotions can be both positive and negative. Like a movie got involved with controversy people will go and watch this movie out of curiosity so that they can justify the controversy. This is an example of negative promotions. Another example can be about photolab. It got involved in a controversy/ negative review that the user's personal data was leaked. Before this controversy photolab had few downloads in playstore but after this controversy Photolab became popular and there's know millions download of Photolab in Playstore. This is negative promotion.

## Software Project Management:

* Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.
* Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Here are 6 activities of a project manager.

1. Proposal writing: A documentation of customer's problem and how we'll solve it.
2. Project planning & scheduling: Planning how to execute the project like what method, models to use and scheduling the phases of project
3. Project costing: Here we think about the revenue from the project. How much the customer will pay and what are my costing for this project. If costing is greater than payment then we can request to increase the payment or drop the project.
4. Project monitoring and reviews : Monitoring the developers and reviewing their work daily. This is to ensure we can deliver the product on time.
5. Personnel selection and evaluation: Project manager usually selects personnel to work on their project. So ideally he seeks skilled staff with good experience for his project. But sometimes managers have to settle for less than ideal project team, like less experienced or new employees. So, he has to evaluate them to know they can do the project or not. A project manager may need to select less than ideal project team for the reasons below

The project budget may not cover highly paid experienced personnel

Staff with appropriate experience may not be available

The company wants to improve skill of their employees

1. Report writing and presentations: Project managers are usually responsible for reporting on the project to both the client and contractor organization’s. They have to write concise, coherent documents that abstract critical information from detailed project reports. They must be able to present this information during progress reviews. Consequently, if you are a project manager, you have to be able to communicate effectively both orally and in writing.

**Project planning:**

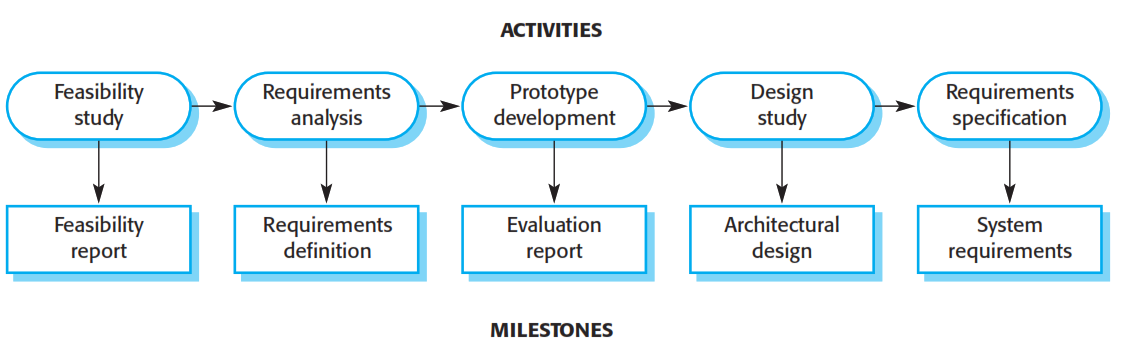
* Probably the most time-consuming project management activity.
* Continuous activity from initial concept through   
  to system delivery. Plans must be regularly revised as new information becomes available.
* Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

**Project plan structure:**

* Introduction : As the name suggest, it'll have the intro of the project.
* Project organization : Organizations involved in a project.
* Risk analysis : Defining or analysis types of problem we can face in a project.
* Hardware & software resource requirements : Identifying resource requirements.
* Work breakdown : Breaking down the whole project in small parts
* Project schedule : Defining the completion time of each part of the project
* Monitoring & reporting mechanism : Mainly done by team lead/project manager daily.

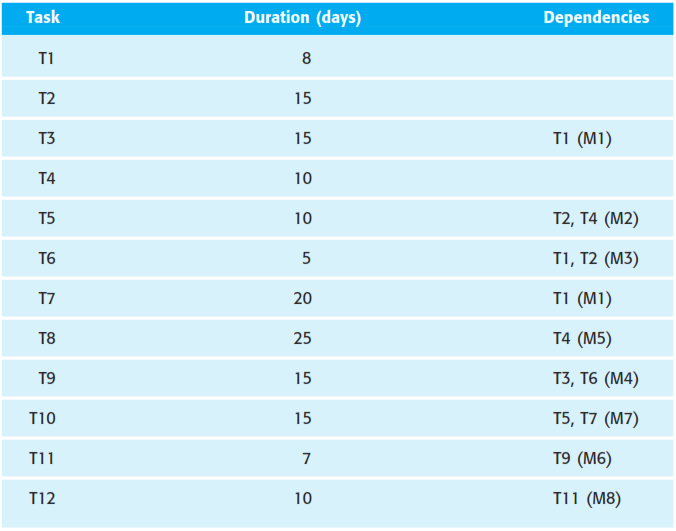
**Milestones and Deliverables:**

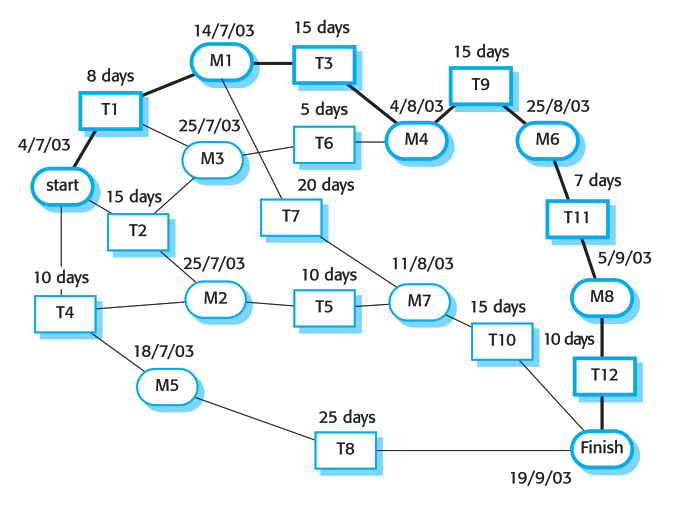
* *Milestones* are the end-point of a process activity.
* *Deliverables* are project results delivered to customers.



* Graphical notations used to illustrate the project schedule.
* Show project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
* Activity charts show task dependencies and the critical path.
* Bar charts show schedule against calendar time.

**Task durations and dependencies:**







**T2**

**T4**

**T1**

**T3**

**M1**

**T5**

**M2**

**T6**

**M3**

**8 days**

**15 days**

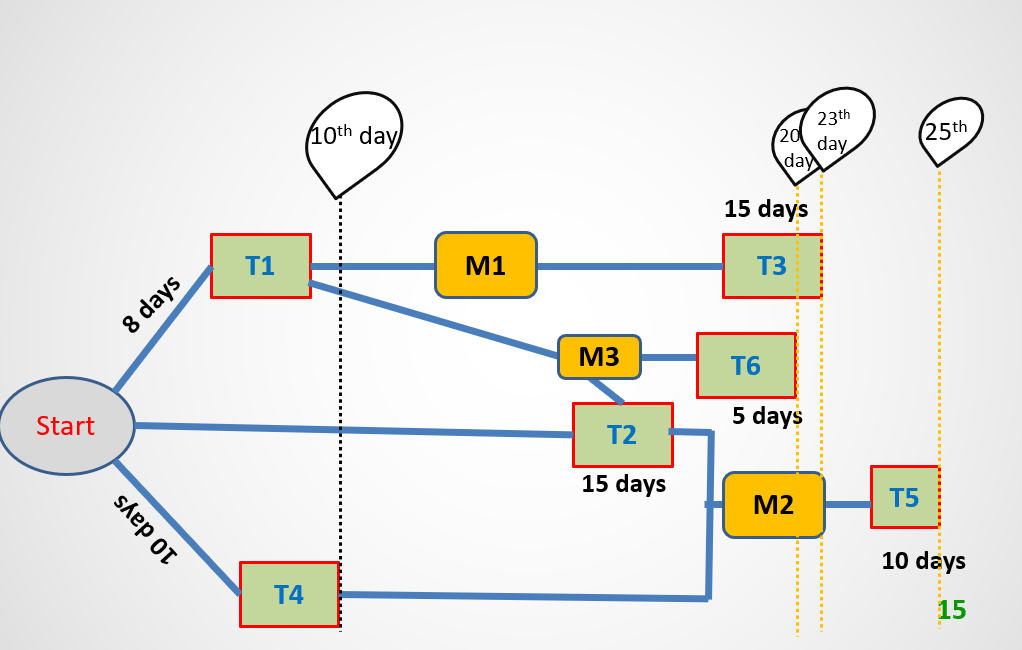
**15 days**

**5 days**

**10 days**

**10 days**





**Risk management:**

* Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
* A risk is a probability that some adverse circumstance will occur
  + Project risks affect schedule or resources;
  + Product risks affect the quality or performance of the software being developed;
  + Business risks affect the organisation developing or procuring the software.

**The risk management process**:

* **Risk identification**
  + Identify project, product and business risks;
* **Risk analysis**
  + Assess the likelihood and consequences of these risks;
* **Risk planning**
  + Draw up plans to avoid or minimise the effects of the risk;
* **Risk monitoring**
  + Monitor the risks throughout the project;



**The risk management process:**

* **Risk identification**

There are at least six types of risk that can arise:

* **Technology risks:** Risks that derive from the software or hardware technologies that are used to develop the system.
* **People risks:** Risks that are associated with the people in the development team.
* **Organisational risks:** Risks that derive from the organisational environment where the software is being developed.
* **Tools risks:** Risks that derive from the CASE tools and other support software used to develop the system.
* **Requirements risks**: Risks that derive from changes to the customer requirements and the process of managing the requirements change.
* **Estimation risks:** Risks that derive from the management estimates of the system characteristics and the resources required to build the system**.**

**The probability of the risk might be assessed as very low (<10%), low**

**(10–25%), moderate (25-50%), high (50–75%) or very high (>75%).**

**Chapter: 06-07:**

Requirements Engineering

* The requirements for a system are the descriptions of the services provided by the system and its operational constraints.
* The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering (RE).**
* **User requirements**
  + Statements in **natural language** plus **diagrams of the services** the system provides and its operational constraints.
* **System requirements**
  + A structured document setting out detailed descriptions of the system’s functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.
* **Functional requirements**
  + Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
* **Non-functional requirements**
  + constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
* **Domain requirements**
  + Requirements that come from the application domain of the system and that reflect characteristics of that domain.

**Software requirements document:**

IEEE standard suggests the following structure for requirements documents:

**1. Introduction**

1.1 Purpose of the requirements document

1.2 Scope of the product

1.3 Deﬁnitions, acronyms and abbreviations

1.4 References

1.5 Overview of the remainder of the document

**2. General description**

2.1 Product perspective

2.2 Product functions

2.3 User characteristics

2.4 General constraints

2.5 Assumptions and dependencies

**3. Speciﬁc requirements** cover functional, non-functional and interface requirements.

**4. Appendices**

**5. Index**

**Feasibility studies:**

A **feasibility study** decides whether or not the proposed system is worthwhile.

A feasibility study is a short, focused study that aims to answer a number of

questions:

1. Does the system contribute to the overall objectives of the organization?

2. Can the system be implemented using current technology and within given cost and schedule constraints?

3. Can the system be integrated with other systems which are already in place?

**Feasibility study implementation:**

Based on information assessment (what is required), information collection and report writing

Questions for people in the organization

* What if the system wasn’t implemented?
* What are current process problems?
* How will the proposed system help?
* Is new technology needed? What skills?
* What will be the integration problems?
* What facilities must be supported by the system?
* What is the risk associated with cost and schedule?
* What are the potential disadvantages/advantages?
* Are there legal issues?
* Are there issues linked with the fact that this is an offshore project?

**Requirements validation:**

* **Validity Check:**

The system should provide the functions which best support the customer’s needs.

* **Consistency Check:**

requirements in the document should not conflict.

* **Completeness Check:**

The requirements document should include requirements, which define all functions and constraints required by the customer.

* **Realism Check:**

The requirements can be implemented with given available budget and technology.

* **Verifiability**:

The delivered system should meet specified requirement.

**Lecture 11: chapter 8,System models**

**What is model?**

* Model is a simplification of reality.
* A **model** is an abstraction of some aspect of an existing or planned system.
* Blueprint of the actual system.
* Specify the structural and behavior of the system.
* Templates for designing the system.

**System Modeling**

* Modeling is a **central part** of all the activities that lead to the development of good software.
* We build models to
  + To Communicate the desired structure and behavior of our system.
  + To visualize and control the systems architecture.
  + To better understanding the system we are building often exposing opportunities for simplification and reuse.
  + To manage risk.
* Modeling is a proven and well accepted engineering tech.

**Why Model ?**

* Analyse the problem-domain
  + simplify reality
  + capture requirements
  + visualize the system in its entirety
  + specify the structure and/or behaviour of the system
* Design the solution
  + document the solution - in terms of its structure, behaviour, etc.
* Modeling achieves four aims:
  + Helps you to **visualize a system** as you want it to be.
  + Permits you to **specify the structure or behavior of a system.**
  + Gives you a **template that guides** **you in constructing** a system.
  + **Documents the decisions** you have made.
* You build models of complex systems because you cannot comprehend such a system in its entirety.
* You build models to better understand the system you are developing.

**Principles of Modeling**

**First principle of modeling**:

The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.

**Second principle of modeling:**

Every model may be expressed at different levels of precision

**Third principle of modeling**:

The best models are connected to reality.

**Fourth principle of modeling**:

No single model is sufficient. Every non-trivial system is best approached through a small set of nearly independent models.

**What are the advantages of creating a model?**

**Advantages:**

- Provides standard for software development.  
- Reducing of costs to develop diagrams of UML using supporting tools.  
- Development time is reduced.  
- The past faced issues by the developers are no longer exists.  
- Has large visual elements to construct and easy to follow.

**System Modeling :**

* **System modeling** is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
* System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
* System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

**Examples of the types of system models that you might create during the analysis process are:**

* 1. A data- flow model Data-flow models show how data is processed at different stages in the system.
  2. A composition model A composition or aggregation model shows how entities in the system are composed of other entities.
  3. An architectural model Architectural models show the principal sub-systems that make up a system.
  4. A classification model Object class/inheritance diagrams show how entities have common characteristics.
  5. A stimulus-response model A stimulus-response model, or state transition diagram, shows how the system reacts to internal and external events.

**Unified Modeling Language (UML):**

* The Unified Modelling Language (UML) is an industry standard for object oriented design notation.
* The **UML** stands for **Unified modeling language**.
* The UML is a language for
  + - * Visualizing
      * Specifying
      * Constructing
      * Documenting

the artifacts of a software-intensive system.

**Characteristics of UML:**

The UML has the following features:

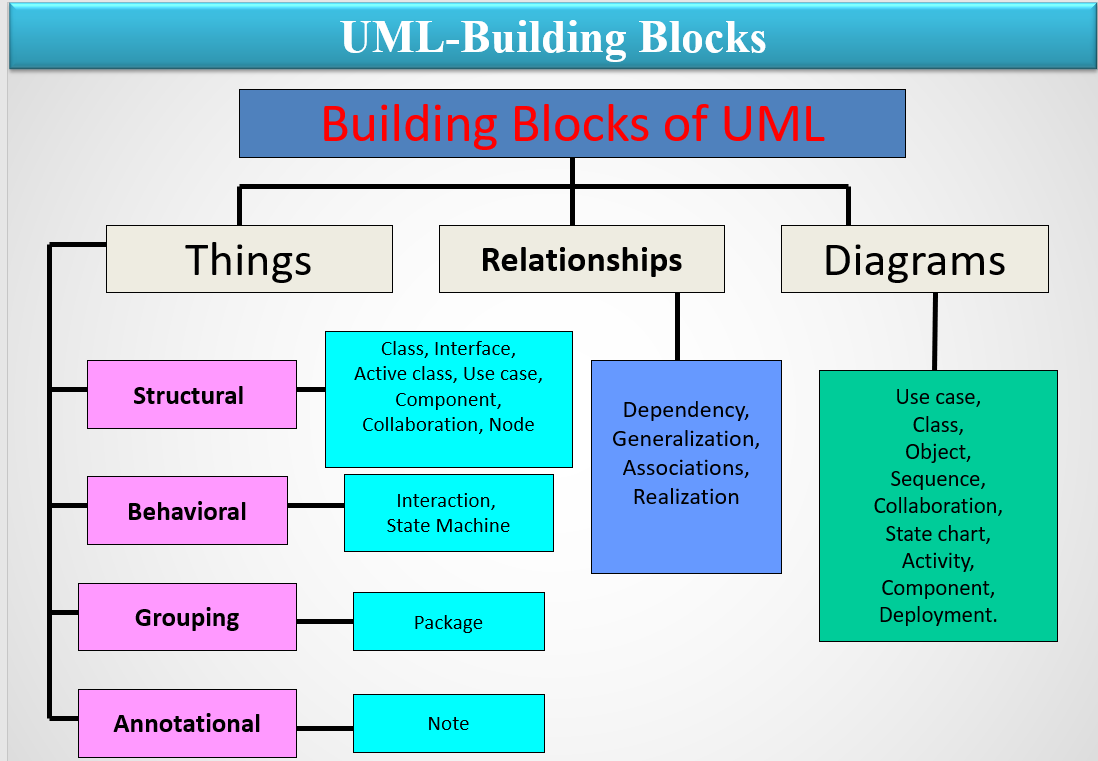
* It is a generalized modeling language.
* It is distinct from other programming languages like C++, Python, etc.
* It is interrelated to object-oriented analysis and design.
* It is used to visualize the workflow of the system.
* It is a pictorial language, used to generate powerful modeling artifacts.

**Where Can the UML Be Used?**

The UML is intended primarily for software-intensive systems. It has been used effectively for such domains as

* + Enterprise information systems
  + Banking and financial services
  + Telecommunications
  + Transportation
  + Defence/aerospace
  + Retail
  + Medical electronics
  + Scientific
  + Distributed Web-based services

The UML is not limited to modeling software. In fact, it is expressive enough to model non-software systems, such as workflow in the legal system, the structure and behavior of a patient healthcare system, software engineering in aircraft combat systems, and the design of hardware.



**Structural Things**

* Structural things are the **nouns of UML models**. These are mostly static parts of a model, representing elements that are either conceptual or physical.

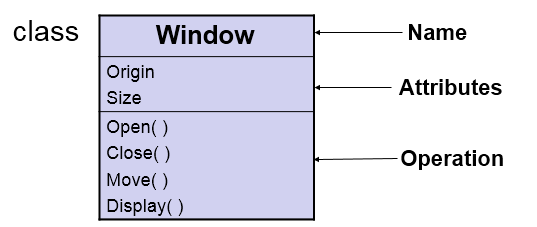
**There are 7 kinds of Structural things:**

* Class
* Interface
* Collaboration
* Use case
* Actor
* Component
* Node

**(1)Class:-**

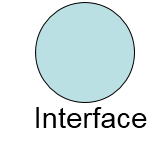
**A class is a description of a set of objects that share the same attributes, operations, relationships and semantics.**

**A class implements one or more interfaces.**



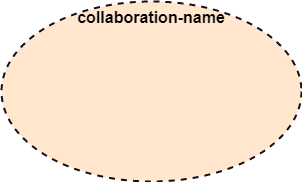
**(2)** Interface:-

* Interface is a collection of operations that specify a service of a class or component.
* An interface describes the externally visible behavior of that element.
* An interface defines a set of operation specifications but never a set of operation implementations.
* Graphically, an interface is represented as a circle with its name.



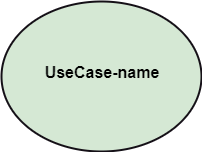
(3) Collaboration:-

* A collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior.
* Therefore, collaboration have structural as well as behavioral dimensions.
* Graphically a collaboration is represented as an ellipse with dashed lines, usually including only its name.



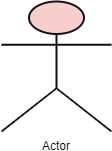
(4) Use case:-

* A Use case is a description of set of sequence of actions that a system performs, which results a value to a particular actor.
* A use case is used to structure the behavioral things in a model.
* A use case is realized by a collaboration.



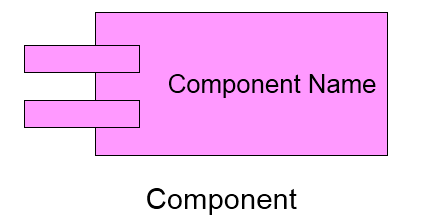
(5) Actor:-

* Actor: It comes under the use case diagrams. It is an object that interacts with the system, for example, a user.



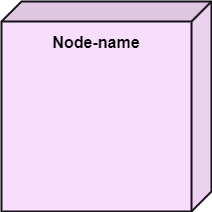
(6) Component:-

* A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
* A component represents the physical packaging of logical elements such as classes, interfaces and collaborations.



(7) Node:-

* A node is physical element that exists at run-time and represents a computational resource, generally having some memory and processing capability.
* A set of components may reside on a node and may also migrate from node to node.



Now,

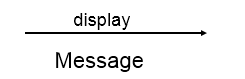
* There are two types of behavioral things

1. Interaction

2. State machine.

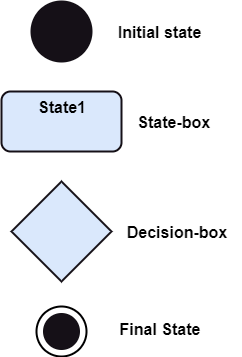
1. Interaction:-

* An Interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to perform a specific purpose.
* An interaction involves a number of other elements, including messages, action sequences and links. (connection between objects.)



(2) State machine:-

* A State machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.
* A State machine involves a number of other elements, including states, transitions( the flow from state to state), events (things that trigger a transition), and activities ( the response to a transition).



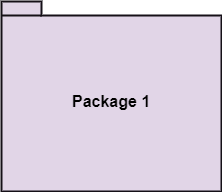
Now,

**UML: Grouping Things**

* Grouping things are the organizational parts of UML models.
* In all, there is one primary kind of grouping thing, namely, Package.

Package:-

* A Package is a general-purpose mechanism for organizing elements into groups.
* Structural things, behavioral things and even other grouping things that may be placed in a package.
* Unlike component (which exist at run time), package is purely conceptual (meaning that it exist only at development time).



**Relationships in the UML**

There are four kinds of relationships in the UML.

1. Dependency

2. Association

3. Generalization

4. Realization

These relationships are the basic relational building blocks of the UML. These are used to write well-formed models.

**DEPENDENCY:-**

* A Dependency is a semantic relationship between two things in which a change to one thing( the independent thing) may affect the semantics of the other thing (the dependent thing).

****

**ASSOCIATION:-**

* A set of links that associates the entities to the UML model. It tells how many elements are actually taking part in forming that relationship.
* It is denoted by a dotted line with arrowheads on both sides to describe the relationship with the element on both sides.



**GENERALIZATION:-**

* A generalization is a specialization/generalization relationship in which objects of the specialized elements (the child) are substitutable for objects of the generalized elements (the parent).
* In this way, the child shares the structure and the behavior of the parent.
* Represented as a solid line with a hollow arrow head pointing to the parent.



Now from here : See :Lecture 11 slide from 28th slide